Roach X

To: Distribution

From: Gary C. Dixon

Subject: Changes to convert_date_to_binary_

Date: January 13, 1975

The segment
>udd>Multics>lib>e>convert_date_to_binary_
contains an improved version of the Multics time conversion program. The program is upwards compatible with the old program, except for the unusual cases documented below. In addition, many new features have been added. If you encounter any other differences between the old and new versions, or if you have comments on the changes, please contact GDixon.PDO on the MIT Multics. Because the old program is installed in the Standard Service System along with such commands as date_time and memo, the new program must be initiated *explicitly* to be used, preferably early in your start_up.ec.

New Features

1) Negative offsets may be used (eg, -4 days, -3 months). The order in which offsets are applied can affect the resultant clock value. For example, does
   10/1 -1 day +1 month
   produce a clock value for 10/30/74 or 10/31/74? The answer is 10/31/74, because offsets are applied in the following order:
   day-of-week offset
   year offset
   month offset
   week offset, day offset, hour offset, second offset
   If the application of a month offset would result in a non-existent date (eg, January 31, 1972 +1 month), then the last day of the month is used, taking leap years into account (in this case, February 29, 1972).
2) A day-of-the-week offset value may be specified, instead of a specific day of the week, by using the new form:
   next day-of-the-week

For example,

    [date 10/31 Monday]

returns an error if the next October 31 does not fall on a Monday, but

    [date 10/31 next Monday]

returns the date of the next Monday after October 31. Both a specific day of the week, and a day-of-the-week offset may be given in the same date/time string. Note that, in

    1245. Monday next Tuesday

Monday would normally be interpreted as a day-of-the-week offset since no specific date was given. However, "next Tuesday" overrides Monday in this case.

3) Year offsets may be used (eg, -2 years, 3 years).

4) The abbreviations for the offset values

    second minute hour day week month year

may be specified as:

    sec   min   hr   da   wk   mo   yr

5) A complete date may be specified in a new form:

    year-of-century.month.day

For example, 75.12.31

6) Times of the form: hhmm.m may now include up to seven digits of fractional minutes. For example: 2359.9999999

7) Spaces are no longer required between alphabetic and numeric fields in the date/time string, although they must still be supplied between two numeric fields, unless the second field begins with a plus (+) or minus (-) sign. For example:

    2days4hours10minutes
    1245.17+7hours
    10/17/74Thursday

8) Underscores may be used instead of spaces in the date/time string.

9) PL/I is used to convert the numeric strings to numbers, so numbers may be signed, but may not include any spaces between the sign and digit, or between digits.

10) The names of acceptable time zones are now obtained from a separate data base (currently called time_table_), instead of being coded into convert_date_to_binary_. The current time_table_ includes those zones defined by the old version of convert_date_to_binary_.

11) The new program uses three new subroutine entry points in performing the conversion. These are:

    encode_clock_value_
    encode_clock_value_$offsets
    decode_clock_value_$date_time

These new subroutine entry points may provide an easier-to-use interface which programs can use to perform specific date/time functions (such as, get me a clock value for 7 days from now). The interfaces are described below.

Changed Features

1) The abbreviations hou (for hour) and wee (for week) may no longer be used.

2) A date/time string of the form:
```
     "2400. mm/dd/yy day-of-week"
```
was and still is mapped into a string of the form:
```
     "0000. mm/dd+1/yy day-of-week"
```
since a time of 2400. is technically illegal but is often used. The new version requires that mm/dd+1/yy fall on the specified day-of-week. Thus, the command:
```
     date_time 2401. 10/15/74 wed
```
returns
```
     10/16/74  0001.0 edt Wed
```
as one would expect. The old program required that mm/dd/yy fall on the specified day-of-the-week.

3) The string
```
     1245.10 /17/74
```
used to be interpreted as
```
     1245.0 10/17/74
```
but is now in error.

Examples

1) The last day of this month can be printed by:
```
     date [month]/1 1 month -1 day
```
2) Yesterday
```
     date -1 day
```
3) Five hours ago
```
     time -5 hours
```
4) Election day
```
     date 10/31 next Monday +1 day
```

Name: encode_clock_value_

This procedure computes a clock value from absolute date/time specifications, or from an input clock value and date/time offset specifications. A Multics clock value is a number of micro-seconds from January 1, 1901 0000.0, Greenwich Mean Time (GMT).

Entry: encode_clock_value_$encode_clock_value_

This entry point creates a Multics clock value from absolute date/time specifications. An absolute date is a month number (1-12), day number (1-31), and year number (1901-1999). An absolute time is an hour number (0-23), minute number (0-59), and second number (0-59), a number of micro-seconds, and one of the time zones listed in time_table_$zones, or a null character string to specify the current time zone (sys_info$time_zone). All dates and times must be valid (eg, 2/29/73 is not a valid date, and 24:00:00 is not a valid time). Also, a day-of-week number (1=Mon, ..., 7=Sun) may be specified. If the day-of-week computed from the date/time specifications does not equal the specified day-of-week, a conversion error is returned.

Usage

```
     dcl  encode_clock_value_  entry  (fixed bin,  fixed bin,
fixed bin,  fixed bin,  fixed bin,  fixed bin,  fixed bin(71),
fixed bin, char(3), fixed bin(71), fixed bin(35));

     call encode_clock_value_ (month, day, year, hour, minute,
second, micro_second, day_of_week, zone, clock, code);
```

1) month	is a month number. (1 = January, 12 = December) (In)
2) day	is a day number. (In)
3) year	is a year number. (1901 <= year <= 1999) (In)
4) hour	is an hour number. (0 to 23, 0 = midnight, 12 = noon) (In)
5) minute	is a minute number. (0 to 59) (In)
6) second	is a number of seconds. (0 to 59) (In)
7) micro_second	is a number of micro-seconds. (In)
8) day_of_week	is a day-of-the-week. (In) (0 = ignore the day-of-the-week) (1 = Mon, ... 7 = Sun)
9) zone	is the time zone in which the times are expressed, or is a null character string to indicate the current time zone. (In) If null, the current time zone is output. (Out)
10) clock	is the encoded clock value. (Out)
11) code	is a status code. (Out)

Entry: encode_clock_value_$offsets

This entry point creates a new Multics clock value by adjusting an input clock value to a specified day-of-week and then adding relative date/time offsets. If the day-of-week is zero, no day-of-week adjusting is performed. The relative date/time values include a year offset, month offset, day offset, hour offset, minute offset, second offset, and micro-second offset. Any of these values may be positive, zero (no offset from input clock value) or negative (backwards offset from input clock value). In addition, an input time zone is specified which may be any of the time zones in time_table_$zones, or may be a null string indicating the current time zone (sys_info$time_zone). The order of applying offsets can affect the resultant clock value. In all cases, the order required by convert_date_to_binary_ has been used. The order is as follows:

1) decode the input clock value into absolute date/time values specified in terms of the input time zone. This zone may affect the day-of-week represented by the input clock value, and hence, may affect any day-of-week offset adjustment.
2) apply any day-of-week offset by adding days to the absolute date until the day-of-week represented by the decoded clock value equals the specified day-of-week.
3) apply any year offset to the decoded clock value.
4) apply any month offset to the decoded clock value. If applying the month offset results in a non-existent date (eg, "Jan 31 3 months" would yield April 31), then use the last day of the month (taking leap years into account) instead.
5) apply the day offset, hour offset, minute offset, second offset, and micro-second offset.
6) encode the resultant absolute date/time specification into the output clock value.

Usage

```
     dcl encode_clock_value_$offsets entry (fixed bin(71),
fixed bin, fixed bin, fixed bin, fixed bin, fixed bin, fixed bin,
fixed bin(71), fixed bin, char(3), fixed bin(71), fixed bin(35));

     call encode_clock_value_$offsets (in_clock, month_offset,
day_offset, year_offset, hour_offset, minute_offset,
second_offset, micro_second_offset, day_of_week_offset, zone,
out_clock, code);
```

1) in_clock	is the clock value to which the offsets are to be applied. (In)
2) month_offset	is an offset, in months. (In)
3) day_offset	is an offset, in days. (In)
4) year_offset	is an offset, in years. (In)
5) hour_offset	is an offset, in hours. (In)
6) minute_offset	is an offset, in minutes. (In)

```
7) second_offset     is an offset, in seconds. (In)
8) micro_second_offset is an offset, in micro-seconds. (In)
9) day_of_week_offset is a day-of-the-week offset. (In)
                        (0 = no day-of-the-week offset.)
                        (1 = Mon, ..., 7 = Sun)
10) zone               is a time zone to be  used  in  applying  the
                       offsets,  or a null character string. (In) If
                       null, the current time zone is output.  (Out)
11) out_clock          is the resultant clock value. (Out)
12) code               is an error code. (Out)
```

END

<u>Name</u>: decode_clock_value_

Given a Multics standard calander clock value, decode_clock_value_ will decode this value into a date and time value.

<u>Entry</u>: decode_clock_value_$decode_clock_value_

This entry point returns the month, day of the month, the year, the time of day, and the day of the week represented by a Multics standard calendar clock value. In addition, the current time zone, used in the calculation, is returned.

<u>Usage</u>

```
     declare decode_clock_value_ entry (fixed bin(71), fixed bin,
fixed bin,  fixed bin,  fixed bin(71),  fixed bin,  char(3),
fixed bin(35));

     call  decode_clock_value_  (clock,  month,  day, year, time,
day_of_week, zone);
```

1) clock is the clock value to be decode. It must represent a date within the 20th Century. (In)
2) month is a month number (January = 1, December = 12) (Out)
3) day   is the number of a day of the month. (Out)
4) year   is the number of a year (e.g, 1973). (Out)
5) time   is the time of day, in micro-seconds since midnight. (Output)
6) day_of_week is the number of a day of the week (Monday = 1, Sunday = 7). (Out)
7) zone   is the current time zone, in which the date and time numbers are expressed. (Out)

<u>Entry</u>: decode_clock_value_$date_time

This entry point returns the month, day of the month, the year, the hour of the day, the minute of the hour, the second of the minute, the micro-seconds of the second, and the day of the week represented by a Multics standard calendar clock value. The caller may specify one of the time zones in the time_table_ in which the decoded clock value is to be expressed, or may request that the value be expressed in the current time zone.

<u>Usage</u>

```
     declare decode_clock_value_$date_time entry  (fixed bin(71),
fixed bin, fixed bin, fixed bin, fixed bin, fixed bin, fixed bin,
fixed bin(71), fixed bin, char(3));

     call decode_clock_value_$date_time (clock, month, day, year,
hour, minute, second, micro_second, day_of_week, zone, code);
```

1) clock is the clock value to be decoded. (In)
2) month is a month number (January = 1, December = 12). (Out)
3) day is the number of a day of the month. (Out)
4) year is the number of a year. (Out)
5) hour is the number of an hour of the day (midnight = 0, noon = 12, 11 PM = 23). (Out)
6) minute is the numbar of a minute of the hour. (Out)
7) second is the number of a second of the hour. (Out)
8) micro_second
   is the number of micro-seconds in excess of a second. (Out)
9) day_of_week
   is the number of the day of the week. (Out)
10) zone is the character string abbreviation of one of the time zones in the time_table_. The decoded clock value is to be expressed in this time zone. (In)
   If the zone character string is a blank string, then the clock value is expressed in the current time zone, and the character string abbreviation for that zone is returned. (Out)
11) code is one of the following status codes. (Out)
   0 the clock value was decoded successfully.
   error_table_$unknown_zone
      the time zone specified by the caller was not found in the time_table_.
   gcd_error_table_$bad_clock_value
      the clock value to be decoded did not lie within the 20th Century.

__Note__

If the clock value given to decode_clock_value_$decode_clock_value_ does not lie within the 20th Century, then zero values will be returned for the the month, day, year, time, and day of the week, and a blank time zone will be returned.

END